

# Definicije in funkcije

V prejšnji lekciji smo spoznali, kako v Mathematici manipuliramo izraze s prepisovalnimi pravili. Prepisovalna pravila smo uporabljali z ukazoma `/.` in `//.`. Včasih želimo, da naj se kako prepisovalno pravilo uporabi samodejno in vedno, kadar je to mogoče. To dosežemo z (*globalnimi*) *definicijami*.

## Definicije spremenljivk

V Mathematici obstajata dve vrsti definicij:

`x = e`      `x` naj bo enak trenutni vrednosti `e`

`x := e`      `x` naj bo enak `e`

V prvem primeru se izvede `e` in nato se njegova trenutna vrednost priredi `x`.

V drugem primeru se `e` ne izvede, ampak si Mathematica samo zabeleži, da se vrednost `x` dobi tako, da se pogleda vrednost `e`. Ponazorimo to z nekaj primeri:

```
a = 42
```

```
x = a
```

```
a = 13
```

Koliko je sedaj vrednost `x`?

```
x
```

Če uporabimo `:=` potem se stvari obrnejo drugače:

```
a = 100
```

```
x := a
```

```
x
```

```
a = 200
```

```
x
```

Splošno pravilo, ki se obnese v večini primerov je, da se za nastavljanje vrednosti spremenljivk uporabi `=` in ne `:=`.

## Brisanje definicije

Definicijo spremenljivke ali funkcije zberemo z ukazom `ClearAll`.

```
x = 20
```

```
ClearAll [x]
```

```
x
```

## Definicije funkcij

Definicije funkcij v Mathematici so pravzaprav prepisovalna pravila, ki jih Mathematica uporabi vedno, ko je to mogoče. Tudi pri definiciji funkcij lahko uporabimo `=` ali `:=`. Funkcijo se definira takole:

`f[p] = e`      Vzorec `f[p]` se vedno prepíše v trenutno vrednost `e`

```
f[p] := e      Vzorec f[p] se vedno prepíše v e
```

```
f[p]; q := e      Vzorec f[p] se vedno prepíše v e, če velja pogoj q.
```

Splošno pravilo, ki se obnese v večini primerov je, da se za definicijo funkcije uporabi `:=` in ne `=`.

Tipičen primer, ko je bolje uporabiti `=` namesto `:=`, se zgodi, ko je vrednost funkcije enaka kompliciranemu izrazu, ki ga zna Mathematica poenostaviti. Na primer:

```
g[a_] = FullSimplify [Sum [  $\frac{1}{a + k^2}$ , {k, 0,  $\infty$ } ]]
Plot [g[x], {x, -2, 2}]
```

Če bi namesto `=` uporabili `:=` bi se izvedel `FullSimplify` vsakič, ko bi računali vrednost funkcije `g`. To bi bilo dokaj zamudno (poskusi!), še posebej pri risanju grafa, ko je treba izračunati vrednosti za veliko število točk.

Definicija dane funkcije  $f$  lahko sestoji iz več pravil. Na primer, takole definiramo Fibonaccijevo zaporedje:

```
f[0] := 1
f[1] := 1
f[n_] := f[n - 1] + f[n - 2]
f[10]
```

### Notacija $f[e]$ , $f@e$ in $e//f$

Funkcijo `f` uporabimo na izrazu `e` tako, da napišemo `f[e]`. Mathematica pozna še dve ekvivalentni notaciji za `f[e]`:

```
f[e]      uporabi funkcijo f na izrazu e (običajna notacija)
f @ e    uporabi funkcijo f na izrazu e (notacija brez oklepajev)
e // f   uporabi funkcijo f na izrazu e (postfiksna notacija)
```

Postfiksno notacijo se ponavadi uporablja za ukaze `Simplify`, `FullSimplify`, in razne prikaze kot so `MatrixForm`, `ColumnForm` in `TableForm`.

```
Sin[x]^2 - Cos[x]^2 // Simplify
r = {{1, 0, 0}, {0, Cos[ $\alpha$ ], -Sin[ $\alpha$ ]}, {0, Sin[ $\alpha$ ], Cos[ $\alpha$ ]}}
r // MatrixForm
r // TableForm
Column[{12, 13, 14, 15, 16}]
```

### Memoizacija

Zgornja definicija Fibonaccijevega zaporedja je slaba, ker deluje zelo počasi (poskusi izračunati `f[100]`!). Naslednji trik nam omogoča, da jo zelo pospešimo:

```
ClearAll [f]
f[0] := 1
f[1] := 1
f[n_] := f[n] = f[n - 1] + f[n - 2]
```

Pojasnimo, kako to deluje. Vrednost izraza `x=e` je enaka trenutni vrednosti `e`, kako v programskem jeziku C. Tretjo vrstico v definiciji `f` lahko razumemo takole: ko izračunaš vrednost `f[m]` po tretjem pravilu, hkrati še definiraj, da je `f[m]` enak vrednosti, ki si jo pravkar izračunal. Se pravi, da si *zapomni* vsako vrednost, ki smo jo izračunali.

```
f[10]
```

Trenutno definicijo funkcije `f` vidimo z ukazom `??f`:

```
?? f
```